9/A
dlg
7/30/03

IN THE
UNITED STATES PATENT AND TRADEMARK OFFICE

APPLICANT: Andrew Wolfe

SERIAL NO.: 09/715,701

FILING DATE: November 16, 2000

TITLE: A Superscalar 3D Graphics Engine

EXAMINER: Sajous, Wesner

ART UNIT : 2676

ATTY. DKT. NO: PA1748

COMMISSIONER FOR PATENTS
P.O. BOX 1450
ALEXANDRIA, VIRGINIA 22313-1450

AMENDMENT AND RESPONSE

In response to the non-final Office Action mailed March 20, 2003, please amend the application as follows.

In the Specification:

Please replace the second paragraph on page 6 with the following.

Referring now to FIG. 1, and more specifically, a first stage in a pipeline is a world transform stage 105, in which the graphics engine converts the vertices and normals of the triangle from the real world object space, which may be different for each object in the scene, to the shared world space, which is space shared by all of the objects to be rendered in the entire scene. This transform consists of a matrix-vector multiplication for each vertex and each normal. In a second stage of the pipeline, a lighting stage 110, the graphics engine takes the triangle's color and surface normal(s) and computes the effect of one or more light sources. The result is a color at each vertex. At the next stage in the pipeline, a view transform stage 115, the graphics engine converts the vertices from the world space to a camera space, with the viewer (or camera) at the center or origin and all vertices then mapped relative from that origin. Additionally, in

the view transform stage 115, the graphics engine applies a matrix-vector multiplication to each vertex calculated for the camera space.

Please replace the third paragraph starting on page 6 with the following.

As further shown in FIG. 1, the next stage in the pipeline is a projection transform stage 120. At the projection transform stage 120, the graphics engine maps the vertices for the camera space to the actual view space. This includes the perspective transformation from 3D to 2D. Accordingly, at this point in the pipeline, the vertices are effectively two-dimensional to which perspective effects (i.e., depth foreshortening) have been applied. Accordingly, the third (z) coordinate is only needed to indicate the relative front-to-back ordering of the vertices when the objects are rendered or drawn within the view space. Like the other two transform stages in the pipeline, the projection transform stage requires the application of a matrix-vector multiplication per each vertex. In a clipping stage 125, the graphics engine clips the triangles or primitives to fit within the view space. Accordingly, the triangles or primitives which lie entirely off the side of the screen or behind the viewer are removed. Meanwhile, triangles or primitives which are only partially out of bounds are trimmed. This generally requires splitting the resulting polygon into additional multiple triangles or primitives and processing each one of these additional triangles or primitives separately. Finally, in a rasterization stage 130, the graphics engine converts those triangles to be displayed within the view space into pixels and computes the color value to be displayed at each pixel. This includes visible-surface determination (dropping pixels which are obscured by a triangle closer to the viewer), texture mapping, and alpha blending (transparency effects).

Please replace the second paragraph starting on page 15 with the following.

Graphics primitives can have several source regions as well as a destination region; but, for simplicity, consider a case where there is one of each. In this case and referring to FIG. 4c, a previously dispatched (and currently processing/executing) primitive *D* 402 will have a set of source pixel locations or a source operand surrounded by a bounding box and a set of destination pixel locations or a destination operand which is also surrounded by a bounding box. A candidate (or new primitive which has yet to be

processed/executed) P 404 will also have a set of source pixel locations or a source operand surrounded by a bounding box and a set of destination pixel locations or a destination operand which is also surrounded by a bounding box. In a preferred embodiment, in order to determine whether or not the candidate primitive P depends on the previously dispatched primitive D, the function *depend*(P,D) is computed. Now, if S_P is the source region of P and D_P is the destination region of P and, furthermore, S_D and D_D are the source and destination regions of D, the dependency between the two can be determined by the following equation:

Please replace the third paragraph on page 23 with the following.

A preferred embodiment of hardware used to compute dependencies is shown in FIG. 7. As shown in FIG. 7, the bounding box coordinates for destination regions and source regions of a new primitive are driven onto vertical buses for each operand. In the example shown, the new primitive has a single destination region 701 and two source regions 702 and 703. Additionally, the bounding box coordinates for destination regions which are stored in each valid reservation station are driven onto horizontal bus lines. At intersections which correspond to potential hazards, bounding box coordinate overlap comparators 704 implement the *depend* function described earlier. For example, a first comparator 704a may compare the destination region (i.e., destination bounding box) of a previously dispatched primitive with the destination of the new primitive, and subsequently generate a first resultant bit. Similarly, a second comparator 704b may compare the source region (i.e., source bounding box) of the previously dispatched primitive with the destination region of the new primitive, and generate a second resultant bit. Furthermore, a third comparator 704c may compare the destination region of the previously dispatched primitive with the source region of the new primitive, and generate a third resultant bit. Although only three comparators are used in this example, alternatively, a different number of comparators may be utilized. Subsequently, a logic OR gate 706 receives the first, second, and third resultant bits and performs a logic OR operation in order to determine whether any dependencies exist between the previously dispatched primitive and the new primitive. A dependence vector is thus calculated by computing whether or not the destination regions or source regions of the new primitive

overlap with the destination regions of each (previously dispatched) primitive which is currently executing. Bit *k* in the dependence vector is set if the new primitive must wait for the primitive stored in destination reservation station *k* to complete, where *k* is the position in the destination reservation station where the conflicting primitives' destination region coordinates are stored (i.e., destination reservation station 1, destination reservation station 2, etc.). This dependence vector is stored in the candidate buffer reserved for the new primitive.

At the same time, the issue unit is testing the existing issue candidates to see if any are ready to be issued. If any of the accelerators/rasterizers are available, then the issue unit tests all of the dependence vectors in the candidate buffers. If any valid candidate buffer contains a dependence vector of all zeros, then the primitive in the candidate buffer can be passed to the available graphics accelerator/rasterizer for processing on the next cycle. In a preferred embodiment, if more than one primitive in the candidate buffer has no dependency conflicts then that entry which corresponds with the earlier primitive or the primitive which has been in the candidate buffer the longest is the one which is selected for processing. At the next clock cycle, the primitive from the candidate buffer is issued by transferring the primitive data and the tag to the available accelerator and clearing the valid bit in order to free that space up in the candidate buffer so a new primitive can be passed from the fetch unit to the issue unit.

Please replace the second paragraph on page 24 with the following

As explained earlier, as the accelerators/rasterizers complete execution and processing of a primitive, the tag corresponding with that primitive is returned to the issue unit. The issue unit will then clear the valid bit for the entry in the destination reservation station which corresponds with that primitive to free up that space in the destination reservation station. The issue unit will also use the tag corresponding with the completed primitive to clear valid bits in the source reservation stations in order to make these spaces available for other primitives. Finally, the tag is decoded and the corresponding bit in any dependence vector in the candidate buffer is cleared. This removes all information associated with the completed primitive from the reservation stations and clears any dependencies for pending primitives associated with that